Spitfire ’91, a WPF Shmup

Victor Gasior and Jaxx Woods

Abstract

Spitfire ’91 is a retro inspired Shoot ’em Up game (or shmup) where you control a small airplane fighting enemy aircraft. The target audience for this project is fans of retro schups like EDF or UN Squadron.

1. Introduction

Spitfire ’91 is a fun, fast paced, retro inspired Shoot ’em Up game made in WPF / C Sharp. In Spitfire ’91 the player controls a modern military fighter jet and fights enemy jets (repersented by colored squares). The game is designed to emulate an SNES (Super Nintendo Entertainment System) era Shoot ’em Up game with its own art, design, sounds, and mechanics. There is one stage where the players can fight enemies until they die in order to try to get a higher and higher high scores. The target audience for this game will be Shoot ’em Up enthusiasts and fans of retro styled video games in general. The audience should first and foremost enjoy the game, while also tailoring the experience to resemble a retro Shoot ’em Up without making it difficult for fans of modern video games to enjoy. There are not many Shoot ’em Up games produced any more so we are appealing to a niche audience starved of content.

1.1. Background

Shoot em’ Ups rely on a player controlled vehicle that can move around the screen and shoot enemies that are trying to attack the player. The player obtains points by avoiding and defeating enemies. The game ends once they reach the end of the level / stage or loses do to being hit by enemies too many times, some Shoot ’em Ups are endless where a player goes for a highscore, like Galaga.

The reason we decided to undertake this project is that Shoot ’em Ups are both fun to play, and seem interesting to design.

1.1.1. Important Terminology.

- Shump - Common abbreviation for Shoot ’em Up
- High Score - A goal of a Shoot ’em Up game, players try to earn higher scores.
- Collision - How the game detects whether or not a game object has toutched another game object.

1.2. Impacts

This game should bring enjoyment to the audience and provide a free, modern alternative to old retro Shoot ’em Up games that can be hard to play due to old hardware and games not being ported to modern systems.

1.3. Challenges

Core game functions like handling enemy, projectile, and player collision in WPF. Normally we would use a dedicated game engine like Godot or Unity, but due to the requirements of this project, we must make these functions ourself in WPF.

2. Scope

This game contains one playable level with multiple enemy variations that can spawn. This level is infinite and goes until the player loses, like Galaga. There will be a win and lose state, as well as a connected lose state screen on loss. There will be a main menu screen the user accesses first before starting the main game. There will be a high score tracker which displays and saves the highest scores to add replay value. Current Stretch Goals are as follows: multiple levels with different backgrounds, a working upgrade shop and currency system, and music. Music is currently completed, as it was very quick and easy. With the upgrade shop players can use some of their score (the curreny) to purchase upgrades for their ship.

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Play Game	user	High	1
2	Check Highscore	user	Med	2
3	Pause Game	user	Easy	3

TABLE 1. SAMPLE USE CASE TABLE

2.1. Requirements

We developed our requirements based on what we think will be the most fun for the user to experience. The requirements both make it so the game is operational (i.e. game actually works, core elements, etc.), while also trying to make the user experience as great as possible (i.e. sound and visual effect feedback).

2.1.1. Functional.

- User needs to be able to control the character to avoid enemies and shoot enemies.
- User needs to be able to earn points by eliminating enemies.
- User needs to be able to view their current score.
- Game needs to spawn enemies for the player to both avoid and shoot.
- User needs to be able to lose.
- There should be visual effects and sound effects to enhance the game experience.
- There should be a scrolling background to give the user the impression of flying

2.1.2. Non-Functional.

- High scores will be persistenty stored between runs
- Custom keybinidng (such as changing movement from arrow keys to WASD)
- Enemy randomization for replayability

2.2. Use Cases

1.

Use Case Number: 1
Use Case Name: Playing the Game
Description: A user wants to play the game.

1) User opens the application.
2) User hits play.
3) User plays the game until loss.

Termination Outcome: The user's score is saved, highest score is updated if this score is the new highest score, which is displayed on the main menu.

Use Case Number: 2
Use Case Name: Check Highscore
Description: A user wants to check the highscore of the local game.

1) User opens the application.

Termination Outcome: The user looks at the score on the main menu. If they are midgame they can look at the top right corner of the screen where the current score is displayed.

Use Case Number: 3
Use Case Name: Pausing the Game
Description: A user wants to pause the game during use

1) User is currently playing the game.
2) User pauses the game using the p key

Termination Outcome: The user keeps the game paused for as long as they want, unpausing if they wish by hitting p again

2.3. Interface

Since we did not get to making the Interface elements such as the Game Over Screen and Main Menu in this version of the project, included are the mockups for them in the interface section.

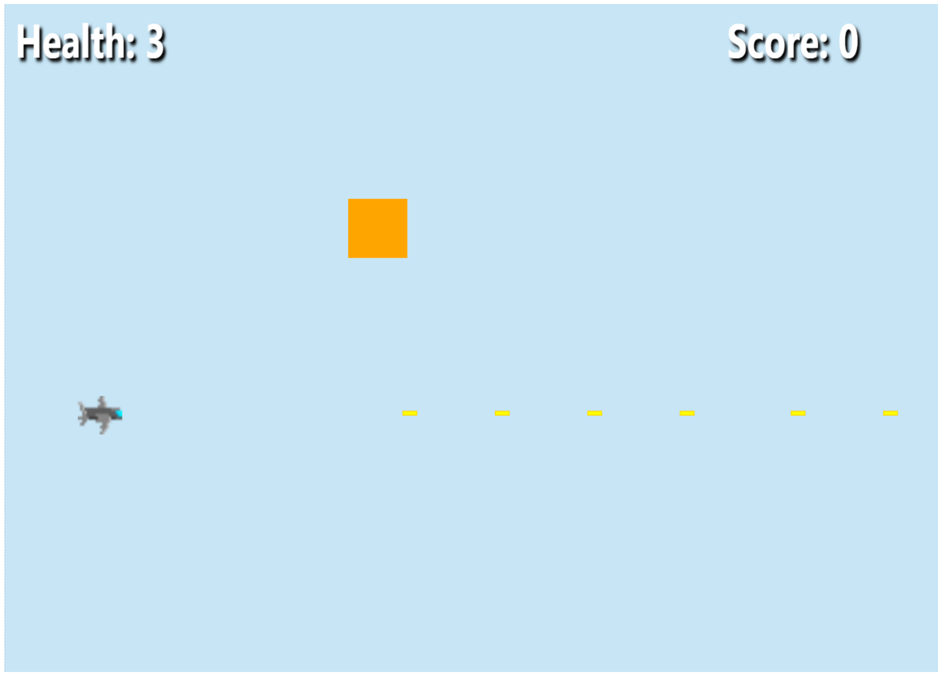

Figure 1. This displays how the game looks during play

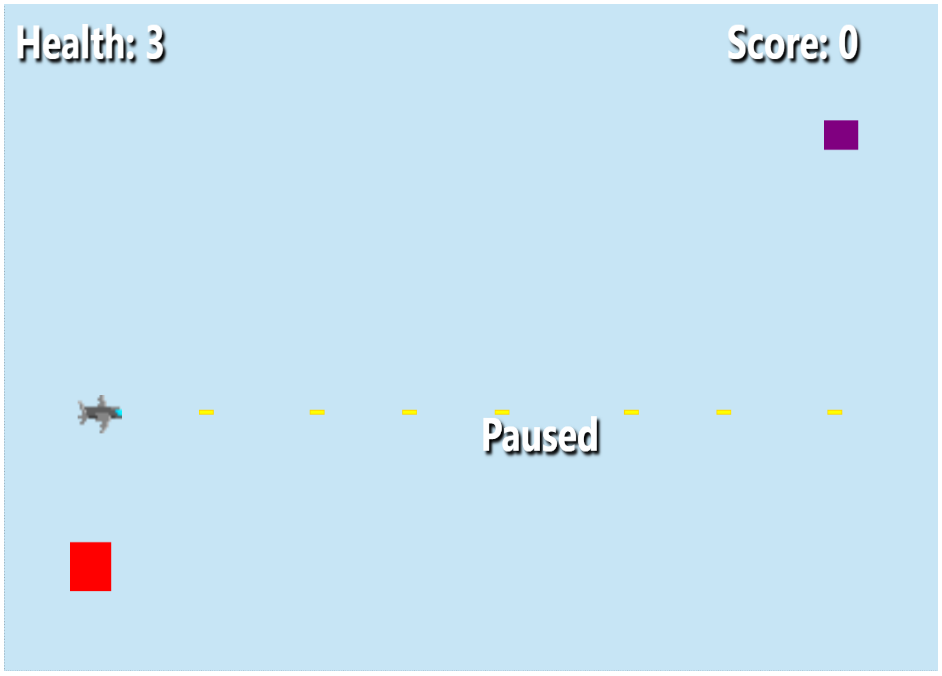

Figure 2. This displays how the game looks when it is paused

3. Project Timeline

We have already made a demoable prototype that we showed in class, but we have yet to develop the full featured product. The steps we took or will take to create the project will go as follows:

1) We have made a game engine that will control several important functions such as: Movement control, enemy spawning, and collision detection.
2) We have made the movement controls and projectile firing.

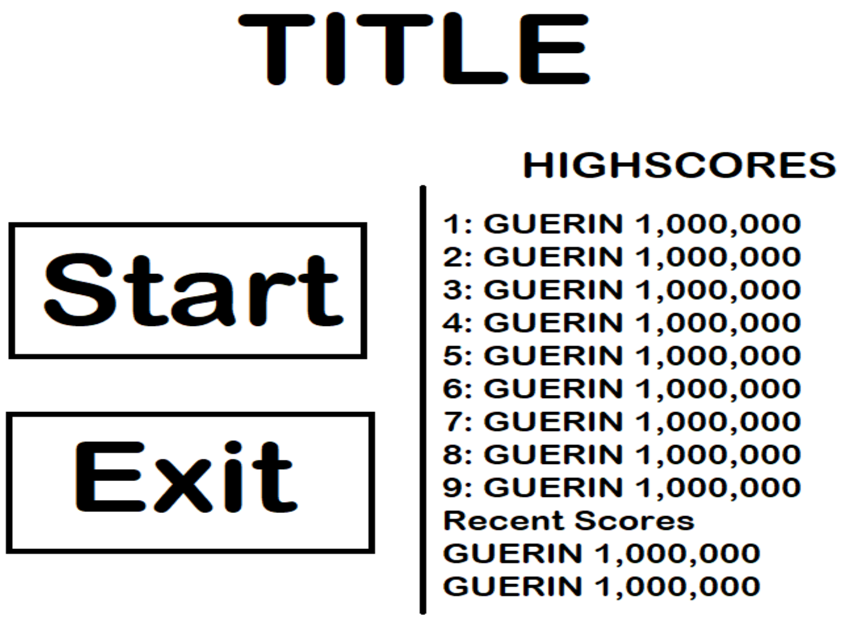

Figure 3. This displays an idea for how the menu screen will look

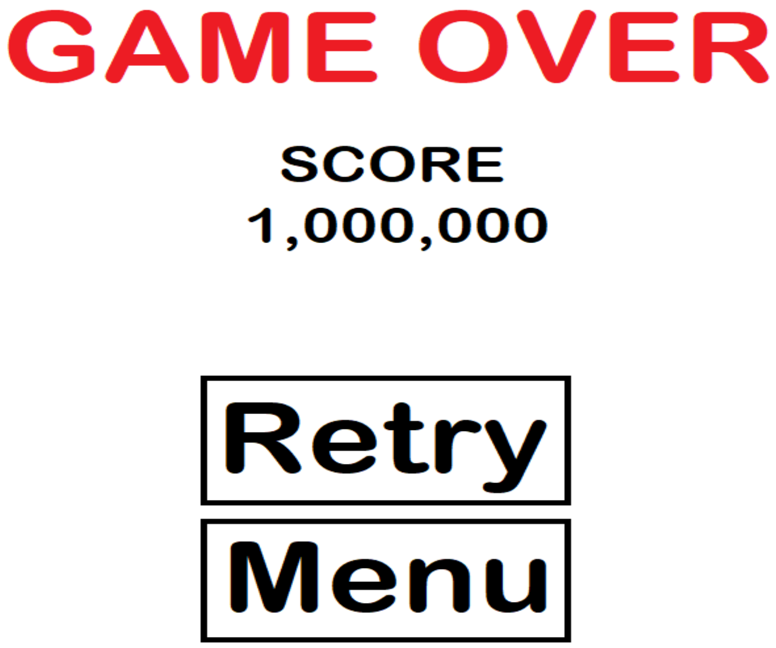

Figure 4. This displays an idea for how the gameover screen will look

3) We have made the spawning of enemies and allow several different versions of enemies to be selected. This includes making an basic enemy object and its derrived children.
4) We have made the Pause Function
5) We will creare the system of collision detection. This also controls the losing of health and gaining of points as certain collisions are detected.
6) We will create a Game Over state and display the appropriate state for a game over.
7) Will we reate a main menu with High Score Board.

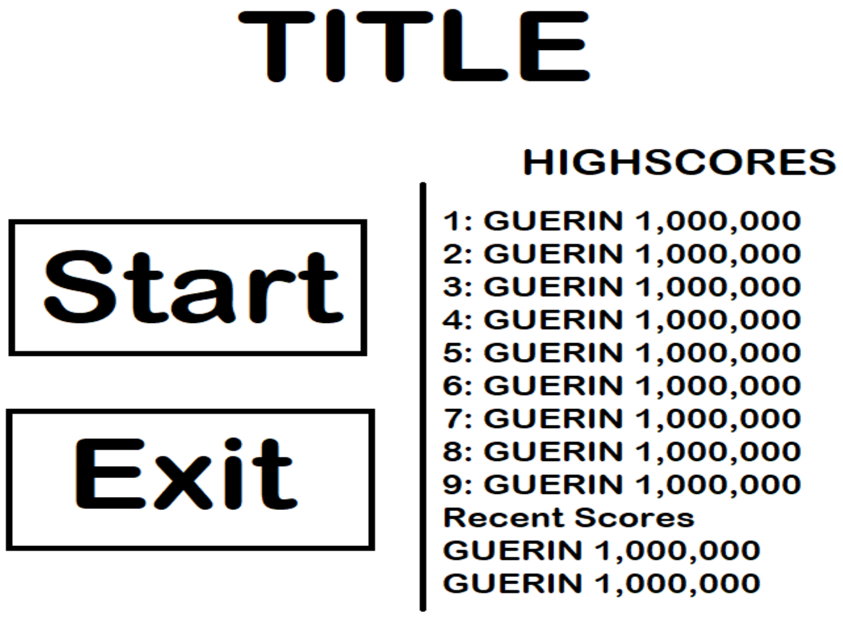

Figure 5. This displays an idea for how the menu screen will look

8) We will polish Audio and Visual presentation.

3.1. Important Dates

- 4/28/22 - In class demo
- 4/29/22 - Final documentation for the semester

4. Project Structure

The project switches between three main windows, that open and close as appropriate. The Main Menu has the controls for starting and exiting the game as well as displaying the highest scores. The GameMain is what controls the game and is what the player will spend most of their time in. This window is switches to GameOverWindow upon the player losing the game. The GameOverWindow allows the player to go straight to either the GameMain or MainMenu window, as well as displaying the score the player just achieved. The game currently has three different sizes of enemies, each will reward more points like targets (i.e. the smaller the target, the more points it is worth).

4.1. UML Outline

6 I'n this figure you can see the UML for our program. The meat of the program is in the GameEngine class, that contains several important functions that are called by both the game's main window GameMain and the GameOverWindow. The Bullet class is derived from the bullet template (Template Design Pattern) and provides the fireBullet function the Rectangle it needs to add to the screen. The spawnEnemy function needs an Enemy class, including EnemyBasic and it's two children (Factory Design Pattern). The three windows are designed to be swapped between when certain functions are called. The enemies use a Factory Design Pattern because the creational pattern is used to make different variants off of the basic enemy class. The bullet uses a Template Design Pattern because the bullet class is derived from an abstract template that could have multiple variations.

4.2. Design Patterns Used

We are implementing the Factory Pattern and Template Pattern for the project. You can see the factory pattern with the enemy classes, with two children of EnemyBasic being derived from EnemyBasic to create new enemy variants. Similarly, Template pattern is used by the bullet class. Although only one type of bullet was created for the project, it still derives from an abstract Template.

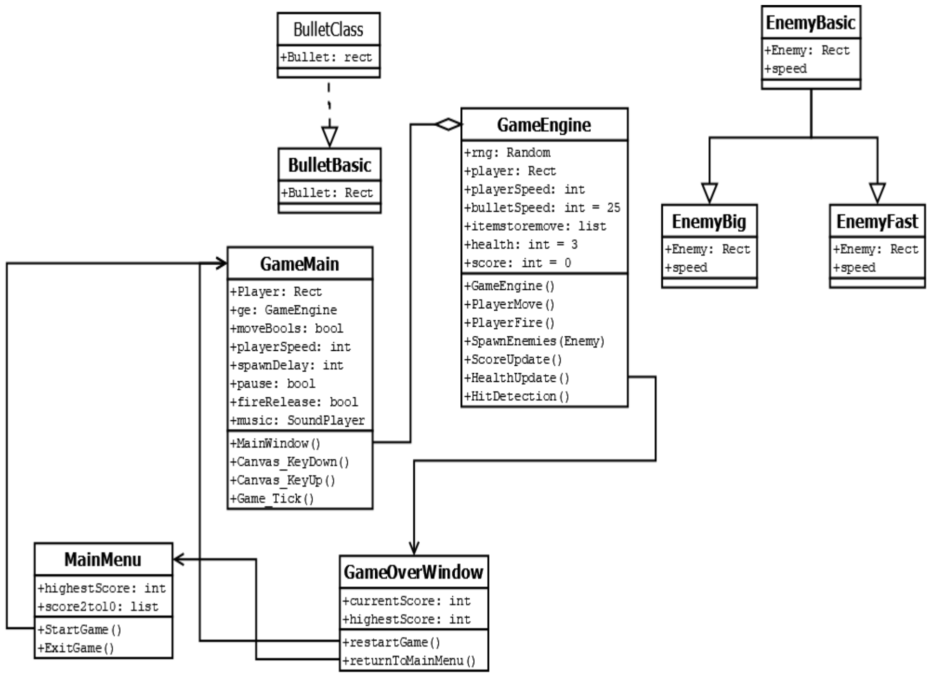

Figure 6. This is the UML for Spitfire ’91

5. Results

Turns out WPF is not really meant for making games, and by not really, we mean not at all. We had to find weird workarounds and use things in strange unintended ways. We worked with the default WPF canvas. Currently we have a game where you can move around the plane and shoot, but not much else. As it currently stands, enemy spawning is working, but their collision detection isn’t. Without enemy collison various game functions like health, scoring, and the game over have no function. We were able to put in music, because that ended up being much simpler to do than we thought it would be.

5.1. Future Work

We are about $\frac{2}{3}^{rds}$ done with the project, but we have run out of the time in the semester. Highest priority is getting collision detection to work, as without collision detection key functions of the game like scoring have no purpose or function. Once that is done we can move to making the Main Menu and Game Over menu, including the saving and displaying of high scores.